ARPA ORDER NO. 189-1

MEMORANDUM
RM-6028-ARPA
JANUARY 1970

EXTENSIONS TO THE PL/I LANGUAGE FOR INTERACTIVE COMPUTER GRAPHICS

R. H. Anderson and D. J. Farber

PREPARED FOR:
ADVANCED RESEARCH PROJECTS AGENCY

The RAND Corporation
SANTA MONICA • CALIFORNIA

ARPA ORDER NO. 189-1

MEMORANDUM
RM-6028-ARPA
JANUARY 1970

EXTENSIONS TO THE PL/I LANGUAGE FOR INTERACTIVE COMPUTER GRAPHICS

R. H. Anderson and D. J. Farber

This research is supported by the Advanced Research Projects Agency under Contract No. DAHC15 67 C 0111. Views or conclusions contained in this study should not be interpreted as representing the official opinion or policy of ARPA.

The RAND Corporation
1700 MAIN ST • SANTA MONICA • CALIFORNIA • 90406

PREFACE

The IBM Conversational Programming System (CPS) was initially installed at The RAND Corporation so that this project could have multi-user on-line access to computer facilities.

The CPS system, as distributed by IBM, is entirely typewriter-oriented, and uses a major subset of the PL/I programming language. This Memorandum discusses a marriage of the CPS system with the Rand Programmer-Oriented Graphics Operation (POGO) system to give the CPS user a significant graphics capability, and to give the POGO user an extensive interactive computational facility. This Memorandum should therefore be of interest to systems programmers and applications programmers who are concerned with the PL/I language, with computer graphics, or with interactive programming.

The work described in this Memorandum is part of a project--sponsored by the Advanced Research Projects Agency--that investigates user-oriented, flexible software for computer graphics.

This research is directly applicable to command and control computer systems that require flexibility, computation, graphic display of data, and user interaction.

SUMMARY

The PL/I language is a powerful, high-level programming tool. The IBM CPS system allows a programmer to create and execute PL/I programs dynamically at an interactive console. The CPS system has been installed at Rand and extended so that the consoles of Rand's Video Graphic System may be used to gain page-at-a-time access to CPS programs and text data.

The PL/I language is text-oriented; it does not contain commands needed for computer graphics (e.g., "display picture p"). However, a powerful graphics capability exists within the Rand POGO system, which allows users to create pictures constructively and to file and retrieve them.

This Memorandum investigates the question: "Could the PL/I language, as embodied in the CPS system, be extended to give the Video Graphic CPS user access to all the graphics facilities of the POGO system?" The Memorandum demonstrates that the answer is "yes"; surprisingly, few extensions to the PL/I syntax are needed to introduce a significant graphics capability within the language. The resulting system should allow a user of a Video Graphic console to construct and use display-oriented systems in an interactive, familiar way.

CONTENTS

I. INTRODUCTION

The field of programming-language design has emphasized the development of languages that are "natural" and easy to use. In many areas, limited successes in this direction have been made--e.g., the languages SNOBOL [1] and BASIC [2]. Graphics, an important and growing area of interactive computing, has witnessed only modest development of such languages. While the *user* of a graphical system is ordinarily insulated from the difficulties of creating and controlling graphical display systems, the *programmer* who wishes to create such systems has not been so lucky. Because of this, the user cannot practically be the creator also. A dual user-programmer effort is usually necessary, therefore, to create computer-graphics packages. In many cases, this is not advantageous.

To alleviate this problem, the authors offer a natural, easy-to-use language for the construction and control of display-oriented programs. The basic path followed is definition by example. Counter to the "normal" system of constructing displays (defining with subroutine calls the location of objects by giving their X,Y coordinates and a set of drawing instructions), the proposed system basically offers the facilities of the POGO system [3], which allows the user to draw on a screen the objects he wishes to display, to label and name the objects, and to define where to display appropriate data. Extending the PL/I [4] programming language allows the user to show different static displays, to fill in values, and to sense the "hitting" of light buttons on the screen. Thus, the designers have married two previously developed ideas (the POGO system and the interactive PL/I), with extensions, to form an interactive graphical PL/I system. (This Memorandum does not describe all of POGO, but only defines some of its major properties.)

The algorithmic language PL/I was chosen as the primary vehicle for man-computer interaction because of certain properties of the language that closely match the needs of an interactive graphical system. These features include:

1) ON-conditions, which allow such external actions as light-pen hits to affect the running program conveniently and asynchronously;
2) A complete set of I/O statements that can be easily extended to meet the needs of an evolving system;
3) A powerful growing base language to program the often complex tasks demanded by current graphic usage.

Realizing the problems in building an interactive incremental PL/I from scratch, the authors used a subset of PL/I called CPS (Conversational Programming System) [5]. This system, distributed by IBM, provides interactive incremental programming in a slightly restricted dialect of PL/I. Having ON-conditions and complete file I/O, it provides the features necessary to build an interactive graphical PL/I.

The following sections of this Memorandum define several proposed extensions to CPS for the creation of a flexible, natural, graphic environment.

II. LANGUAGE EXTENSIONS

To provide a significant graphics capability within an interactive PL/I language, the authors believe the following will suffice: 1) to extend the syntax of the PUT and GET statements; 2) to add one new ON-condition that can be signalled by light-pen or stylus hits; and 3) to provide a command that places the user in a *create display page* mode and that labels the resulting display page. Importantly, these extensions enable the programmer to create display pages by construction rather than by description. This implies that the graphics terminal has a pointing mechanism (a light-pen or stylus) as an input device. (A keyboard and cursor can simulate a pointing device if necessary.) If a programmer has access to a graphics terminal with these capabilities, he can much more easily create interactive programs that use the terminal effectively.

DISCUSSION

The standard PL/I PUT and GET statements allow data to be placed in or received from two sources--a string or a file. (If the user specifies neither, PL/I assumes the standard stream input or output file.) A natural way to incorporate a graphics capability in PL/I is to offer an additional source or recipient of data, a display page. To DECLARE an identifier as a display (analogous to the way files are declared) is more convenient than to add a new statement to the language (e.g., DISPLAY). This instigates the creation of a prototype display page containing fields and areas within which the user may enter or display data.

An important part of the PL/I "philosophy" is modularity; default attributes allow a programmer to write simple programs without knowing all available options and alternatives. If

modularity is to be maintained for an interactive graphical PL/I, a programmer should begin to use the display facilities easily with little or no knowledge of their full capabilities and with no knowledge of "computer graphics programming." A neophyte PL/I programmer can write "PUT DATA (A,B,C)" to obtain a line of output; he should also be able to write "PUT DISPLAY DATA (A,B,C)" to have that same data displayed, without stating exactly how or where it should appear. However, with increased sophistication, the programmer should be able to control the format of his displays.

The power of computer graphics can be given to an inexperienced programmer in a second way--by allowing him to create displays *constructively*, rather than issuing such a command as BOX (X1,Y1,X2,Y2). The POGO software package elegantly illustrates the simplicity of display creation by construction. Much of the flavor and motivation of the graphic extensions the authors propose for an interactive PL/I result from a desire to incorporate the features demonstrated by the POGO system within a high-level, interactive programming language.

Relatively few language extensions are necessary to provide a significant graphics capability within interactive PL/I. The extensions consist of:

1) One new statement, DISPLAY, that signals that the next action will be the creation of a named display page;
2) Additional options in the PUT and GET statements;
3) An additional ON-condition, PUSH, that relates light-pen or stylus actions to asynchronous program responses.

Each of these extensions is considered individually below.

THE DISPLAY STATEMENT

The authors propose that such a statement as

```
LABEL:  DISPLAY;
```

signal that the next operation desired is the creation of a prototype display page named LABEL. The programmer then enters a construction mode in which such facilities as those offered by the Rand POGO system are available to him; e.g., the ability to:

1) Draw (or point to the boundaries of) horizontal, vertical, or arbitrary line segments;
2) Enter character strings and position them dynamically on the display;
3) Construct rectangles, sets of joined lines, and certain geometric figures;
4) Construct and arrange labeled *value lines* upon which the system may display arithmetic or string data;
5) Construct labeled rectangular "special" areas (which might be invisible when displayed) that are sensitive to light-pen or stylus hits or that can be individually addressable sub-displays within a display page.

A display page, therefore, can be referenced as a unit by a label; it consists of constant data (e.g., character strings and line segments) and several addressable sub-elements:

1) *Value lines*, which may have optional alphanumeric labels;
2) *Special areas*, each of which has an alphanumeric label.

All nonlabeled value lines in a display page are consecutively numbered automatically. The alphanumeric labels on value lines will be used for data-directed I/O. For example, if the value of variable XI is to be output in data mode using display page P, then if a value line labeled XI occurs on P, the value will be shown there; if not, by default it will be shown in the next available numbered line within P. (The sections on PUT and GET statement extensions below discuss this in more detail.)

The following conventions facilitate the referencing of labels and areas within a display page:

1) Any value lines that occur within a special area in a display are completely independent of value lines outside that area; their default numbering is independent and their labels are independent. As a corrolary, to reference a value line in an area A of display P, the labels P and A must both be used to refer to that value line; the reference P is not sufficient.
2) Special-area boxes are global on a display; a display page can contain any number of special areas, but they cannot nest or overlap.

After the programmer signals the completion of display construction or modification, the system files the display page under its label for future reference in PUT and GET statements.

THE PUT STATEMENT

Figure 1 graphs the syntax of an extended PUT statement for an interactive graphical PL/I language. In general, the extensions for display PUT values onto a DISPLAY as well as onto the conventional STRING or FILE (or, by default, the file SYSPRINT). Optional phrases are enclosed in square brackets. The notation [(P[,B])] following the word DISPLAY means these options are available:

PUT DISPLAY -- Does not use a labeled display; by default, the information will be placed in some canonical form on the CRT, just as the POGO system offers default output display pages.

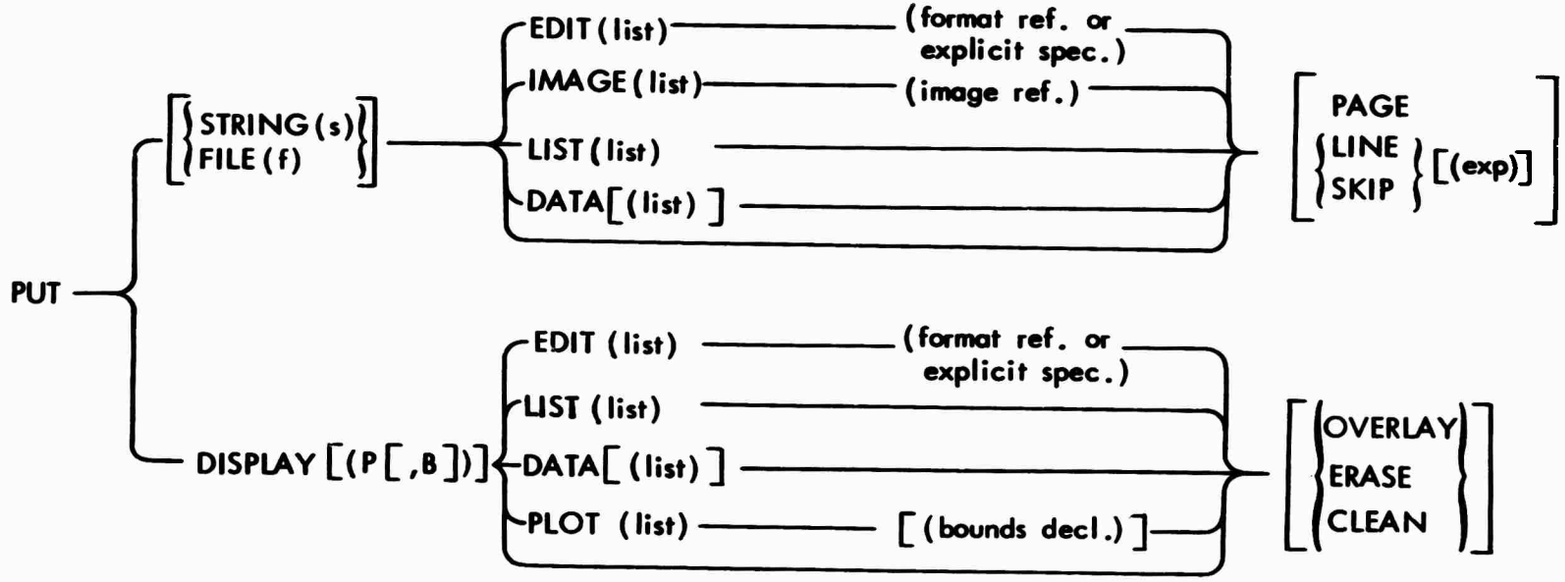

Fig. 1 - Proposed Syntax Graph for an Interactive Graphical PL/I PUT Statement

PUT DISPLAY (P) -- Displays the information using the constructed display page labeled P.

PUT DISPLAY (P,B) -- Displays the information in the area labeled B within the display labeled P.

The options EDIT, LIST, and DATA used with DISPLAY mean much the same as in conventional PL/I; they direct the format of the output values. An EDIT-directed DISPLAY allows the user to specify the number of significant digits shown on the value line and to specify other parameters related to the appearance of the value; a LIST-directed DISPLAY matches the ordering of the listed expressions with the numerically coded value lines used; a DATA-directed DISPLAY matches variable names with the alphanumeric labels on the value lines--any variable without a labeled value line is placed by default on the first numerically coded line available.

The DISPLAY options OVERLAY, ERASE, and CLEAN control the placement of a new display on the CRT. In general, OVERLAY adds information to a display without erasing currently displayed information; it is most useful for superimposing plots in one area. ERASE is the default mode; its operation is rather complex and is explained by example below. CLEAN displays a fresh copy of the prototype display page with no values automatically filled in. (Some automatic fill-in occurs with the ERASE option.)

The following examples trace some possible paths through the graph structure of the PUT statement shown in Fig. 1.

PUT DISPLAY;

Display default display page (probably with some empty value lines).

PUT DISPLAY spec [{OVERLAY / ERASE / CLEAN}] ;

If OVERLAY, the canonic default display is filled in, continuing where the previous fill-in left off;

If CLEAN or ERASE, a fresh copy of the canonic default display is supplied and fill-in begins at the top.

PUT DISPLAY PLOT (list) [(bounds declaration)];

Since the user mentions no explicit prototype display page, the canonic default display for plots is the whole CRT display area. The user may optionally specify bound. If (list) mentions only one vector, the system displays its n values as vertical (y-axis) displacements against n equally-spaced intervals on the x-axis; if (list) names two or more vectors, the system places the first on the x-axis at equally-spaced intervals and plots all others against it, using a unique plotting symbol for each curve.

PUT DISPLAY (P) LIST (V1, V2,, V3) [{OVERLAY / ERASE / CLEAN}] ;

(Assume that all value lines in P have number codes only and that P has one special area labeled A1. Note that the above statement contains two consecutive commas between V2 and V3.)

If the user mentions none of (OVERLAY, ERASE, CLEAN), ERASE is the default.

If P is not the currently displayed picture and the user specifies OVERLAY, the action is the same as if ERASE had been specified.

If P is the currently displayed picture and the user specifies OVERLAY, the picture is unchanged except that the current values of the variables V1, V2, and V3 replace any values shown on value lines 1, 2, and 4.

If the user specifies ERASE, all value lines are erased; then the system fills in lines 1, 2, and 4. If P is the currently displayed picture, area A1 does not change. Under the ERASE option, the system automatically fills in any labeled value lines in display P with the current value of the corresponding variable.

If P is the currently displayed picture and the user specifies CLEAN, the action is the same as if P were not the displayed picture, except that area A1 will not contain information and all value lines except 1, 2, and 4 will be clean.

```
PUT DISPLAY (P, A1) LIST (V1, V2, V3);
```

The system will attempt to place the values V1, V2, and V3 within area A1 of display page P. If area A1 contains value lines, they will be used; otherwise, a default display will be placed in area A1.

If P is the currently displayed picture, the rest of P is unchanged. (If the user specified CLEAN, the system would clear the rest of P of values.)

Comments on implementing the PUT statement:

1) A listed item in a PUT DISPLAY . . . statement may be a vector instead of a scalar; in that case, the system treats it as if the elements of the vector were explicitly enumerated within the list.
2) Display names may be indexed just as statement labels are in PL/I. Therefore, display pages might be labeled, for example, DISP(1) or DISP(2). If display name DISP(J) appeared in a PUT statement, the system would use the value of J to retrieve the correct display page.

THE GET STATEMENT

Figure 2 presents a syntax graph for an augmented GET statement. The additions are very similar to those made to the PUT statement. The figure shows the statement form GET DISPLAY PLOT . . . . This facility would require tablet-stylus or light-pen tracking at program execution time; therefore, a more sophisticated console is necessary. The user may specify input curves in the POGO system; the GET . . . PLOT statement provides a natural access to this capability.

The following examples illustrate some options of the GET statement.

```
GET DISPLAY LIST (V1, V2);
```

Since the user names no particular display, the currently displayed page is used along with the message ENTER DATA; upon receiving the exit signal, the system reads the values entered in the first and second value lines and assigns them to variables V1 and V2.

```
GET DISPLAY (P) LIST (V1, V2);
```

If P is not the current display, this command is equivalent to the pair of commands:

```
PUT DISPLAY (P);
GET DISPLAY LIST (V1, V2);
```

If P is the current display, the system shows the ENTER DATA message; value lines 1 and 2 are intensified (or signalled in some other way). Upon receiving the exit signal, the system reads value lines 1 and 2 and assigns the values to variables V1 and V2.

```
GET DISPLAY (P) DATA (V1, V2);
```

This operates like the above example, except that the system reads the value lines labeled V1 and V2 instead of those numbered 1 and 2.

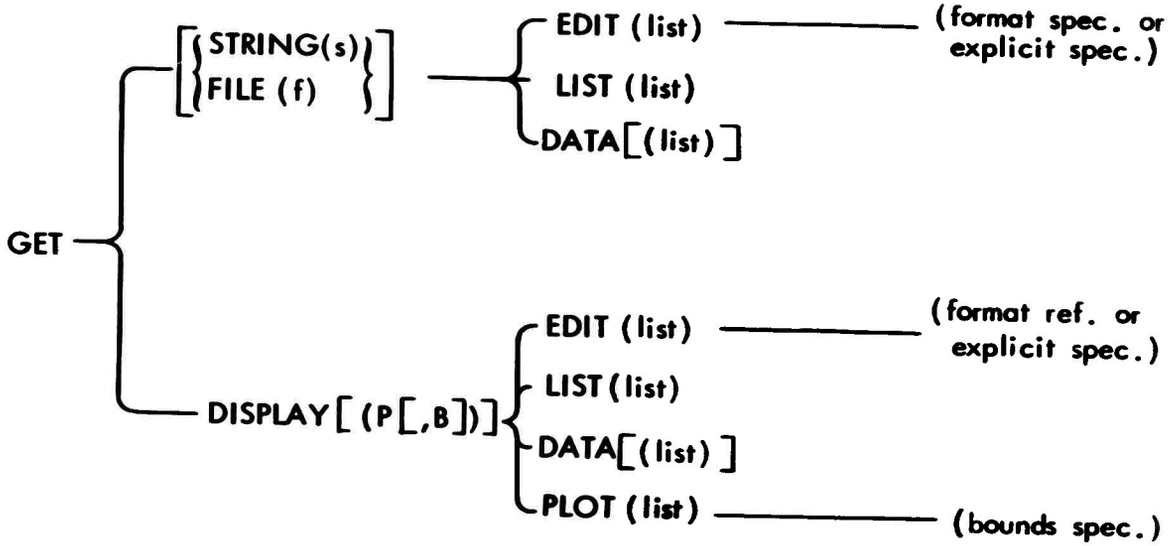

Fig. 2 - Proposed Syntax Graph for an Interactive Graphical PL/I GET Statement

ON-CONDITIONS

ON-conditions in PL/I can associate executable procedures with particular interrupts; the procedures activate whenever the interrupt occurs, not only when the user makes an explicit test for the interrupt. For example, the statement

```
ON OVERFLOW CALL WHOOPS;
```

associates the procedure WHOOPS with the OVERFLOW condition. This mechanism can handle interrupts generated by light-button "hits." To do this, one new ON-condition is necessary: PUSH. The following options should be available:

ON PUSH (P1, B1) on-clause;

After completing the current statement, control transfers to the on-clause if special area B1 of display page P1 is hit; after executing the on-clause, control returns to the place where it was interrupted.

To refer to a light-button name exclusive of the picture in which it occurs, use the following command:

```
ON PUSH ( , B1);
```

ON PUSH (P) ...

This condition activates if the user hits any button in picture P and satisfies no more explicit ON-condition.

Two built-in functions access information about display status. BUTTON always contains the character-string name of the last button hit within the current display (if none is hit, the null string). DISPLAY always contains the character-string name of the current display (if no display is current, the null string).

Handling light-button interrupts with ON-conditions has another potentially useful feature: should true multi-tasking and parallel processing become available, the light-buttons could activate asynchronous tasks. This feature would open interesting new uses of computer graphics in which each displayed light-button *really* represents a process. Thus, for example, several independent processes could operate on a single datum--e.g., placing several simultaneous rotations on a drawn figure. This type of logic could be more easily programmed if multi-tasking becomes available.

III. IMPLEMENTATION

The CPS system provides one interactive PL/I language within which the user might incorporate the extensions suggested in this Memorandum. Rand has implemented CPS on an experimental, video-based graphics system being developed jointly by The RAND Corporation and IBM. The authors intend that the language extensions proposed here be incorporated on that video-CPS system.

BLANK PAGE

REFERENCES

1. Farber, D. J., R. F. Griswold, and I. P. Polonsky, "The SNOBOL3 Programming Language," *Bell System Technical Journal*, Vol. 45, No. 6 (July-August 1966), pp. 895-944.

2. Kemeny, J. G., and T. E. Kurtz, *BASIC Users Manual*, Dartmouth College Computation Center, Hanover, N. H., January 1966.

3. Boehm, B. W., V. R. Lamb, R. L. Mobley, and J. E. Rieber, *POGO: Programmer-Oriented Graphics Operation*, The RAND Corporation, RM-5825-PR, June 1969. (Also, *Proceedings of the Spring Joint Computer Conference*, 1969.)

4. IBM Corporation, "PL/I Reference Manual," Form C28-8201.

5. Sumpter, Edmund J., "Conversational Programming System," Program Order Number 360D-03.4.016, IBM Corporation, Program Information Department (PID), February 1969.

DOCUMENT CONTROL DATA

1. ORIGINATING ACTIVITY The Rand Corporation	2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED 2b. GROUP

3. REPORT TITLE
EXTENSIONS TO THE PL/1 LANGUAGE FOR INTERACTIVE COMPUTER GRAPHICS

4. AUTHOR(S) (Last name, first name, initial)
Anderson, R. H. and D. J. Farber

5. REPORT DATE	6a. TOTAL NO. OF PAGES	6b. NO. OF REFS.
January 1970	24	5

7. CONTRACT OR GRANT NO.	8. ORIGINATOR'S REPORT NO.
DAHC15-67-C-0141	RM-6028-ARPA

9a. AVAILABILITY/LIMITATION NOTICES	9b. SPONSORING AGENCY
DDC 1	Advanced Research Projects Agency

10. ABSTRACT

A proposed combination of the IBM Conversational Programming System (CPS) and the Rand Programmer-Oriented Graphics Operation (POGO). CPS is entirely typewriter oriented and uses a subset of PL/I with the ON-conditions and complete file I/O that are necessary in building an interactive graphical language. The facilities of POGO, which allow the user to draw on a screen the objects he wishes displayed, to label and name objects, and to define where to display appropriate items, are provided by these extensions to the language: (1) one new statement, DISPLAY, signalling the creation of a named display page; (2) additional options in the PUT and GET statements; and (3) an additional ON-condition, PUSH, that relates light-pen actions to asynchronous program responses. The requirement of command and control computer systems for flexibility, computation, graphic display, and user interaction will be served by such a connection of interactive and graphic capabilities.

11. KEY WORDS

Computer Graphics
Video Graphic System
Computer Programming Languages
Man-Machine Interaction
POGO (Programmer Oriented Graphics Operation)